LAPORAN PRAKTIKUM DATA STRUCTURE

Advanced Sort

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033

Oleh :

Muhammad Alif Mujaddid **240605110082**

Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. Praktikum

1. Pendahuluan

1. Shell sort Shell sort menggunakan konsep yang berdasarkan pada insertion sort. Pada insertion sort, setiap item ditandai dengan menyimpannya pada variable temporary kemudian item tersebut dibandingkan dan disisipkan pada sorted group sesuai urutannya. Penyisipan dilakukan dengan menggeser item yang memiliki nilai lebih besar dari marked item sehingga terdapat ruang untuk ditempati oleh marked item. Masalah pada insertion sort tampak ketika nilai yang paling kecil berada pada posisi paling kanan dan harus ditempatkan pada posisi paling kiri di sorted group. Pada kondisi tesebut penyisipan tetap dilakukan dengan menggeser sejumlah item satu ruang kekanan. Performa pada insertion sort tesebut dapat ditingkatkan dengan cara melakukan pergeseran pada jarak tertentu. Hal inilah yang dilakukan pada shell sort. Jarak antar elemen disebut dengan increment value. Misal, pada proses pengurutan 10 elemen array dengan nilai increment 4 (disebut 4-sort), maka tiap subarray yang dibandingkan adalah pada index (0, 4, 8), (1, 5, 9), (2, 6), dan (3, 7). Proses 4-sort menghasilkan array yang hampir terurut. Untuk array yang lebih panjang, dibutuhkan jarak yang lebih besar kemudian secara berulang berkurang hingga bernilai 1 sehingga elemen dapat terurut secara sempurna. Misalkan array dengan 1000 item dapat diurutkan secara bertahap mulai 364-sorted, kemudian 121- sorted, 40-sorted, 13-sorted, 4-sorted, hingga 1-sorted. Deret bilangan yang digunakan untuk membentuk interval ini (pada contoh ini: 364, 121, 40, 13, 4, 1) disebut dengan interval sequence atau gab sequence. Original shell sort menggunakan N/2 untuk menentukan interval sequence, dimana N dimulai dari jumlah elemen dan secara berulang dibagi dengan 2 hingga bernilai 1. Berikut ini listing program untuk shell sort dengan N/2 sequence

```
public void ShellSort () {
  int in, out;
  int temp;
  int h = nElemen / 2;

  while (h > 0) {
    for (out = h; out < nElemen; out++) {
       temp = arr[out];
       in = out;

       while (in > h - 1 &&
              arr[in - h] >= temp) {
          arr[in] = arr[in - h];
          in -= h;
       }
       arr[in] = temp;
    }
    h /= 2;
 }
}
```

Lengkapi listing program tersebut sebagaimana yang pernah Anda kerjakan pada modul 2. Lakukan pengurutan terhadap 8 elemen array. Tambahkan code untuk menampilkan isi array (display();) setelah baris code arr[in] = temp;. Jalankan dan tulis output program tersebut! Jelaskan!

```java
public class ShellSort {
    private int[] arr;
    private int nElemen;
    private int max;

    public ShellSort(int max) {
        this.max = max;
        arr = new int[max];
        nElemen = 0;
    }

    public void insert(int value) {
        arr[nElemen] = value;
        nElemen++;
    }

    public void display() {
        for (int i = 0; i < nElemen; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public void shellSort() {
        int in, out;
        int temp;
        int h = nElemen / 2;

        while (h > 0) {
            for (out = h; out < nElemen; out++) {
                temp = arr[out];
                in = out;
                while (in > h - 1 && arr[in - h] >= temp) {
                    arr[in] = arr[in - h];
                    in -= h;
                }
                arr[in] = temp;

                // tampilkan array setiap kali ada perubahan
                display();
            }
            h /= 2;
        }
    }
```

```java
Run | Debug
public static void main(String[] args) {
    ShellSort data = new ShellSort(max:8);
    data.insert(value:45);
    data.insert(value:12);
    data.insert(value:78);
    data.insert(value:34);
    data.insert(value:23);
    data.insert(value:56);
    data.insert(value:89);
    data.insert(value:11);

    System.out.println(x:"Data sebelum diurutkan:");
    data.display();

    System.out.println(x:"\nProses Shell Sort:");
    data.shellSort();

    System.out.println(x:"\nData setelah diurutkan:")
    data.display();
}
}
```

```
Data sebelum diurutkan:
45 12 78 34 23 56 89 11

Proses Shell Sort:
23 12 78 34 45 56 89 11
23 12 78 34 45 56 89 11
23 12 78 34 45 56 89 11
23 12 78 11 45 56 89 34
23 12 78 11 45 56 89 34
23 11 78 12 45 56 89 34
23 11 45 12 78 56 89 34
23 11 45 12 78 56 89 34
23 11 45 12 78 56 89 34
23 11 45 12 78 34 89 56
11 23 45 12 78 34 89 56
11 23 45 12 78 34 89 56
11 12 23 45 78 34 89 56
11 12 23 45 78 34 89 56
11 12 23 34 45 78 89 56
11 12 23 34 45 78 89 56
11 12 23 34 45 56 78 89

Data setelah diurutkan:
11 12 23 34 45 56 78 89
```

2. N/2 interval sequence dianggap tidak efektif. Knuth telah mengenalkan salah satu bentuk interval sequence yang kini umum digunakan. Knuth's interval sequence menggunakan bentuk kebalikan yang dimulai dari 1 menggunakan persamaan: h = 3 * h + 1

Tabel 6.1 Knuth's interval sequence

H	**3 * h + 1**	**(h - 1) / 3**
1	4	
4	13	1
13	40	4
40	121	13
121	364	40
364	1093	121
1093	3280	364

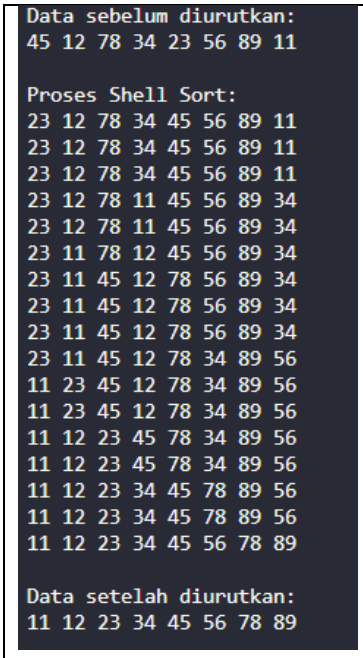
Untuk mengimplementasikan Knuth's interval sequence pada shell sort (listing nomor 1), tunjukkan baris code yang harus dirubah dan tuliskan code-nya! Jelaskan!

```
int h = 1;
while (h < nElemen / 3) {
    h = 3 * h + 1;
}
while (h > 0) {
    for (out = h; out < nElemen; out++) {
        temp = arr[out];
        in = out;
        while (in > h - 1 && arr[in - h] >= temp) {
            arr[in] = arr[in - h];
            in -= h;
        }
        arr[in] = temp;
        display();
    }
    h = (h - 1) / 3;
}
```

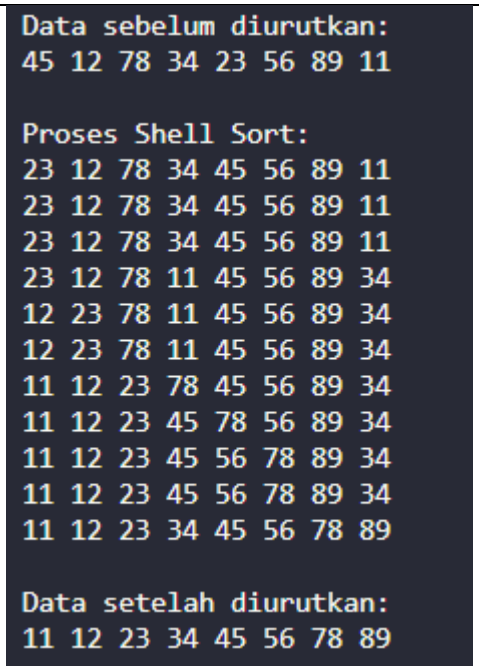
Pada algoritma Shell Sort, cara mengatur jarak (interval) antar elemen yang dibandingkan sangat penting. Kalau biasanya dipakai cara sederhana dengan membagi dua (n/2, n/4, ...), Knuth memberikan cara yang lebih efektif yaitu dengan rumus h = 3 * h + 1. Dengan rumus ini, interval yang dipakai lebih teratur dan membuat proses pengurutan lebih cepat. Misalnya untuk 8 data, interval yang dipakai adalah 4 lalu 1, bukan 4, 2, 1 seperti pada cara lama. Setelah

interval besar selesai dipakai, jarak dikembalikan dengan rumus (h - 1) / 3 sampai akhirnya h = 1. Dengan begitu, data bisa lebih cepat tersusun rapi.

3. Lakukan pengurutan terhadap 8 elemen array sesuai data yang anda gunakan pada soal nomor 1 menggunakan algoritma shell sort dengan interval/ jarak yang dirumuskan oleh Knuth. Jalankan program, bagaimana proses pengurutannya? Bandingkan dengan output program nomor 1. Jelaskan!



Tidak pake knuth



Pake knuth

Proses pengurutan 8 elemen array dengan algoritma Shell Sort menggunakan interval Knuth dimulai dari jarak 4, kemudian dilanjutkan dengan jarak 1. Pada tahap pertama dengan jarak 4, elemen yang dipasangkan dan dibandingkan adalah elemen ke-0 dengan ke-4, ke-1 dengan ke-5, ke-2 dengan ke-6, serta ke-3 dengan ke-7. Dari proses ini terjadi beberapa pertukaran, misalnya 45 ditukar dengan 23, serta 34 ditukar dengan 11. Setelah itu interval diperkecil menjadi 1 sehingga prosesnya sama dengan insertion sort, yang menyusun elemen secara berurutan hingga diperoleh hasil akhir 11, 12, 23, 34, 45, 56, 78, 89. Dibandingkan dengan Shell Sort biasa yang menggunakan interval n/2, metode Knuth lebih efisien karena hanya menggunakan dua tahap (h = 4 dan h = 1), tetapi tetap menghasilkan urutan yang benar. Dengan demikian, algoritma Knuth mampu mempercepat proses sorting karena jumlah perbandingan dan pertukaran elemen lebih sedikit.

4. Partisi Partisi adalah mekanisme yang digunakan pada quick sort. Melakukan partisi data berarti membagi data tersebut menjadi dua bagian berdasarkan nilai batas tertentu. Item yang memiliki nilai lebih kecil dari nilai batas menjadi satu kelompok sedangkan item yang memiliki nilai lebih besar dari nilai batas menjadi satu kelompok yang lain. Batas yang menjadi penentu kelompok data tersebut dikenal sebagai pivot value. Jika terdapat 10 elemen array berikut ini:

60	5	15	45	35	20	25	10	50	30
0	1	2	3	4	5	6	7	8	9

dan diketahui nilai pivot adalah 30. Tuliskan langkah-langkah partisi array tersebut hingga diperoleh array hasil partisi sebagai berikut:

30	5	15	10	25	20	35	45	50	60
0	1	2	3	4	5	6	7	8	9

Jelaskan!

Partisi pada Quick Sort membagi array menjadi dua bagian berdasarkan nilai **pivot**: semua elemen yang lebih kecil dari pivot diletakkan di satu sisi, dan yang lebih besar di sisi lain. Untuk array
[60, 5, 15, 45, 35, 20, 25, 10, 50, 30] dengan **pivot = 30**, kita bisa pakai metode Hoare (dua penunjuk dari kiri dan kanan). Intinya: mulai dari kiri dan kanan, geser penunjuk kiri sampai menemukan elemen ≥ pivot, geser penunjuk kanan sampai menemukan elemen ≤ pivot, lalu tukar kedua elemen itu; ulangi sampai penunjuk kiri melewati penunjuk kanan. Berikut langkah-langkah ringkasnya (menunjukkan swap yang terjadi):

1. Ambil pivot = 30.
2. Bandingkan dari kiri dan kanan: pertama bertemu 60 (kiri) dan 30 (kanan) tukar
   Array [30, 5, 15, 45, 35, 20, 25, 10, 50, 60]
3. Lanjut geser: dari kiri cari ≥30 → ketemu 45 (index 3); dari kanan cari ≤30 ketemu 10 (index7)tukar
   Array [30, 5, 15, 10, 35, 20, 25, 45, 50, 60]
4. Lanjut lagi: dari kiri cari ≥30 ketemu 35 (index 4); dari kanan cari ≤30 ketemu 25 (index 6)tukar
   Array [30, 5, 15, 10, 25, 20, 35, 45, 50, 60]
5. Sekarang penunjuk kiri dan kanan bertemu/berlewati proses partisi selesai.

Hasil akhir partisi (semua < 30 di satu sisi, semua ≥ 30 di sisi lain) adalah: [30, 5, 15, 10, 25, 20, 35, 45, 50, 60].

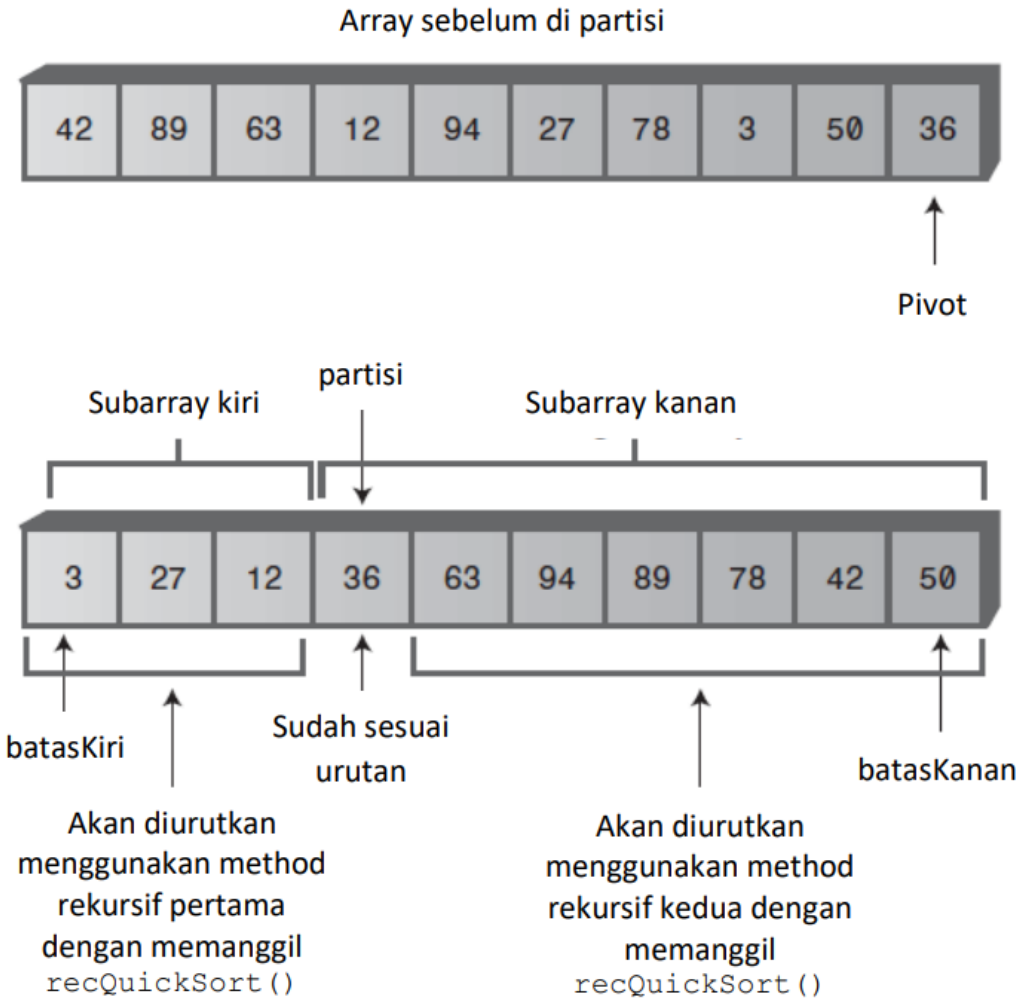
data berdasarkan pivot untuk memudahkan tahap pengurutan berikutnya

secara rekursif.

Berapa kali pertukaran item pada proses partisi array tersebut? 3

Dimana posisi index partisi yang menjadi batas bagian kiri dan kanan?0

5. Quick Sort Quick sort berkerja dengan melakukan patisi sebuah array menjadi dua subarray kemudian memanggil fungsi itu sendiri secara rekursif untuk melakukan quick-sort terhadap tiap subarray tersebut. Berikut ini tiga tahapan dasar dari quick sort: a. Partisi array atau subarray menjadi bagian kiri (kumpulan item yang memiliki nilai lebih kecil) dan bagian kanan (kumpulan item yang memiliki nilai lebih besar). b. Panggil method partisi itu sendiri untuk mengurutkan bagian kiri. c. Panggil method partisi itu sendiri untuk mengurutkan bagian kanan



Gambar 6.1 Pengurutan Subarray secara rekursif

Untuk melakukan partisi maka perlu menentukan nilai pivot. Berikut ini beberapa catatan tentang pivot pada quick sort: - Pivot dipilih dari nilai item yang eksis berada pada array. - Nilai item sebagai pivot dapat dipilih secara acak. Untuk memudahkan, kita dapat memilih item di batas kanan dari subarray yang akan di partisi - Setelah melakukan partisi, jika pivot berada diantara batas kiri dan batas kanan dari subarray, maka kondisi tersebut menunjukkan array telah diurutkan.

Berikut ini listing program untuk *Quick Sort*:

```
public void QuickSort() {
   recQuickSort(0, nElemen - 1);
}
```

```
public void recQuickSort(int batasKiri,
                         int batasKanan) {
   if (batasKanan - batasKiri <= 0) {
       return ;
   } else {
       int pivot = arr[batasKanan];
       int partisi = partitionIt(batasKiri,
                        batasKanan, pivot);
       recQuickSort(batasKiri, partisi - 1);
       recQuickSort(partisi + 1, batasKanan);
   }
}
```

```
public int partitionIt(int batasKiri,
                int batasKanan, int pivot) {
   int indexKiri = batasKiri - 1;
   int indexKanan = batasKanan;
   while (true) {
       while (indexKiri < batasKanan &&
              arr[++indexKiri] < pivot) ;
       while (indexKanan > batasKiri &&
              arr[--indexKanan] > pivot) ;
       if (indexKiri >= indexKanan) {
            break;
       } else {
           swap(indexKiri, indexKanan);
       }
   }
   swap(indexKiri, batasKanan);
   return indexKiri;
}
```

```java
package Praktikum.modul6;

public class QuickSortApp {
    private int[] arr;
    private int nElemen;

    public QuickSortApp(int max) {
        arr = new int[max];
        nElemen = 0;
    }

    public void insert(int value) {
        arr[nElemen] = value;
        nElemen++;
    }

    public void display() {
        for (int i = 0; i < nElemen; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public void swap(int a, int b) {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }

    public void QuickSort() {
        recQuickSort(batasKiri:0, nElemen - 1);
    }
```

```java
    public void recQuickSort(int batasKiri, int batasKanan) {
        if (batasKanan - batasKiri <= 0) {
            return;
        } else {
            int pivot = arr[batasKanan];
            int partisi = partitionIt(batasKiri, batasKanan, pivot);
            recQuickSort(batasKiri, partisi - 1);
            recQuickSort(partisi + 1, batasKanan);
        }
    }

    public int partitionIt(int batasKiri, int batasKanan, int pivot) {
        int indexKiri = batasKiri - 1;
        int indexKanan = batasKanan;
        while (true) {
            while (indexKiri < batasKanan && arr[++indexKiri] < pivot)
                ;
            while (indexKanan > batasKiri && arr[--indexKanan] > pivot)
                ;
            if (indexKiri >= indexKanan) {
                break;
            } else {
                swap(indexKiri, indexKanan);
            }
        }
        swap(indexKiri, batasKanan);
        return indexKiri;
    }
```

```java
    Run | Debug
    public static void main(String[] args) {
        QuickSortApp qs = new QuickSortApp(max:8);

        int[] data = { 60, 5, 15, 45, 35, 20, 25, 10 };
        for (int d : data) {
            qs.insert(d);
        }

        System.out.println(x:"Array sebelum QuickSort:");
        qs.display();

        qs.QuickSort();

        System.out.println(x:"Array setelah QuickSort:");
        qs.display();
    }
}
```

Proses pengurutan dengan algoritma Quick Sort dimulai dengan memilih elemen terakhir sebagai pivot, misalnya angka 45. Pivot ini berfungsi sebagai pemisah antara elemen yang lebih kecil dan yang lebih besar. Setiap elemen dalam array dibandingkan dengan pivot, lalu dipindahkan ke sisi kiri jika nilainya lebih kecil atau dibiarkan di sisi kanan jika lebih besar. Setelah semua dibandingkan, pivot 45 ditempatkan di posisi yang tepat, yaitu indeks ke-5. Dengan begitu, array terbagi menjadi dua bagian: sisi kiri [34, 25, 12, 22, 11] dan sisi kanan [64, 90]. Selanjutnya, algoritma bekerja secara rekursif. Pada bagian kiri [34, 25, 12, 22, 11], dipilih 11 sebagai pivot. Karena 11 merupakan nilai terkecil, posisinya langsung berada di awal array. Kemudian, bagian setelahnya [34, 25, 12, 22] diurutkan lagi dengan pivot 22. Dalam partisi ini, angka yang lebih kecil dari 22 (yaitu 12) diletakkan di sebelah kiri, sementara angka yang lebih besar berada di kanan. Setelah selesai, pivot 22 berada di indeks ke-2, sehingga terbentuk susunan [11, 12, 22, 25, 34] di bagian kiri. Pada sisi kanan [64, 90], proses yang sama dilakukan. Pivot yang dipilih adalah 90. Karena angka 64 sudah lebih kecil, urutannya tidak berubah. Proses rekursif kemudian berhenti karena tiap bagian hanya tersisa satu atau dua elemen yang memang sudah terurut.

Akhirnya, setelah semua langkah rekursif selesai dijalankan, array tersusun sempurna dalam urutan menaik, yaitu:
[11, 12, 22, 25, 34, 45, 64, 90].